**Data and Software Availability**

Before we begin this tutorial, it is important to highlight that this work uses several Shell/C++ scripts and well-established software in the academic field, such as PyMOL 3.0, UCSF Chimera 1.18 and Amber MD 2024. All the files needed to run a new GA or continue where this work left off are located in the Data and Software Availability folder.

As detailed by Santo and Feliciano (2021), the system used for the scoring function is the result of a manual docking performed with the Chimera software. However, it is also possible to use protein-protein docking tools of the user's preference for those who wish to replicate this work. In our case, we used this structure and relaxed it using the traditional energy minimization approach with Amber, as detailed in our first article. This structure can be found in the `relax_system.pdb` file. The `relax_system.pdb` has been divided into two files: `antigen.pdb`, which contains the RBD, and `antibody.pdb`, which contains the GB1.

Initially, we applied the `template.c` script, which is responsible for scanning the space of `antigen.pdb` to generate the template of the contact surface with the mimetic antibody. This template will be used in the scoring function. It is recommended to change the code to scan only the contact area, which reduces the time.

bash

```
g++ -o template.exe template.c
./template.exe
```

After 30 minutes, the code generates a file called `antigen_template.pdb`, which contains the antigen template. In Figure 1, we can see how the antigen template overlays antibody.pdb ( GB1).

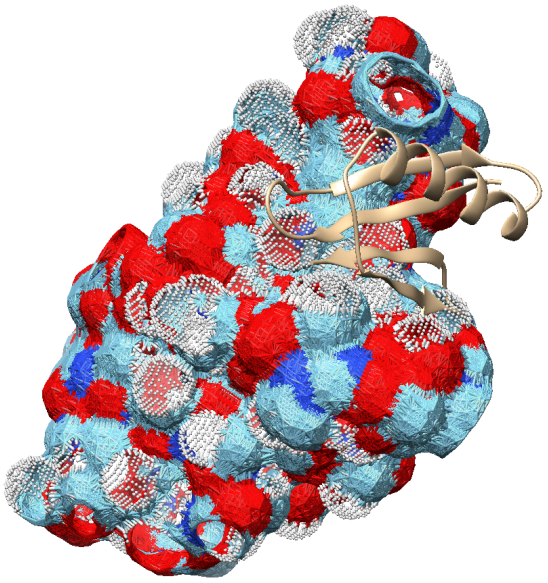

**Figure 1:** Superposition of the GB1-Ref with the antigen template.

Next, we need to compile the `score.c` file, which is responsible for reading the `antigen_template.pdb` and `antibody.pdb` files:

bash
```
g++ score.c -o score.exe
```

Then, we need to prepare the shell scripts responsible for the scoring function:

bash
```
chmod +x score_function.sh
chmod +x pymol.sh
chmod +x pymol1.sh
```

Next, we can execute the `score_function.sh` script to generate a ranking of amino acid residues at the positions of GB1-Ref.

bash

./`score_function.sh`

After execution, which may take approximately one hour, the `score.dat` file will be generated. This file contains the scores for each of the rotamers of every residue in all positions, but we only need the best value. To achieve this, we must compile the code that performs this sorting:

bash

```
g++ -o ranking.exe ranking.c
./ranking.exe score.dat
```

This code will generate the `ranking.dat` file, which contains the rankings of all residues in each position.

Now we can assemble the initial population for running the molecular dynamics of our genetic algorithm. This is done by the `assembly.sh` script.

bash

```
chmod +x assembly.sh
```

```
./assembly.sh
```

After execution, 12 directories will be created, each containing one of the 12 initial mimetic antibodies (MAs), each composed of the highest-ranked amino acid residues. From there, simply perform the 100 ns molecular dynamics simulations. This task is performed by the `genetic.sh` script which, after setting up the environment, uses Amber to perform molecular dynamics simulations and calculates the binding free energy through MMGBSA in each of the directories.

bash

```
chmod +x genetic.sh
```

```
./genetic.sh
```

At the end, in each directory, we will have the `FINAL_RESULTS_MMPBSA.dat` files containing the calculated energies for the mutants. We can then use the `results_mmgbsa.c` script to extract the sequences and energies from each folder. To do this, compile the script with:

bash

```
g++ -o results_mmgbsa.exe results_mmgbsa.c
```

```
./results_mmgbsa.exe
```

The `results_mmgbsa.dat` file will be created with the sequences and energies. Using this data, we can update and reorder the `Population_classification.dat` file.

```
ILE 2 GLN 4 GLU 6 ARG 8 GLU 13 ALA 15 LEU 17 LEU 19 TYR 42 THR 44 GLN 46 SER 49 ILE 51 TYR 53 TRP 55 -75.650
TYR 2 GLN 4 TRP 6 ARG 8 GLU 13 GLU 15 LEU 17 MET 19 TYR 42 GLN 44 GLN 46 SER 49 ILE 51 TYR 53 TRP 55 -68.460
TYR 2 GLN 4 GLU 6 ARG 8 GLU 13 ALA 15 LEU 17 LEU 19 TYR 42 GLN 44 GLN 46 GLN 49 LEU 51 VAL 53 ILE 55 -64.690
TYR 2 GLN 4 GLU 6 ARG 8 GLU 13 ALA 15 LEU 17 LEU 19 TYR 42 GLN 44 GLN 46 SER 49 ILE 51 TYR 53 TRP 55 -62.730
TYR 2 GLN 4 TRP 6 ARG 8 LEU 13 TRP 15 ASN 17 SER 19 TYR 42 GLN 44 GLN 46 GLN 49 LEU 51 VAL 53 ILE 55 -56.690
TYR 2 GLN 4 TRP 6 ARG 8 MET 13 TRP 15 ASN 17 SER 19 TYR 42 GLN 44 GLN 46 GLN 49 ALA 51 VAL 53 MET 55 -55.000
TYR 2 GLN 4 TRP 6 ARG 8 MET 13 TRP 15 TYR 17 SER 19 TYR 42 GLN 44 GLN 46 GLN 49 ALA 51 VAL 53 MET 55 -54.080
TYR 2 GLN 4 GLU 6 ARG 8 MET 13 TRP 15 ASN 17 SER 19 TYR 42 GLN 44 GLN 46 GLN 49 ALA 51 VAL 53 MET 55 -52.920
TYR 2 GLN 4 GLU 6 ARG 8 GLU 13 ALA 15 LEU 17 LEU 19 ARG 42 GLN 44 GLN 46 SER 49 ILE 51 TYR 53 TRP 55 -51.210
TYR 2 GLN 4 GLU 6 ARG 8 GLU 13 ALA 15 LEU 17 LEU 19 TYR 42 GLN 44 GLN 46 GLN 49 LEU 51 TYR 53 ILE 55 -51.060
GLU 2 GLN 4 TYR 6 ARG 8 MET 13 LEU 15 TYR 17 ILE 19 LEU 42 GLN 44 TRP 46 ILE 49 ALA 51 ARG 53 MET 55 -50.490
TYR 2 GLN 4 GLU 6 ARG 8 GLU 13 ALA 15 LEU 17 LEU 19 TYR 42 GLN 44 GLN 46 GLN 49 ALA 51 VAL 53 MET 55 -49.910
ILE 2 GLN 4 ASP 6 ARG 8 ASP 13 ALA 15 LEU 17 LEU 19 TYR 42 GLN 44 GLN 46 SER 49 ILE 51 TYR 53 TRP 55 -49.550
GLN 2 GLN 4 GLN 6 ARG 8 HIS 13 ALA 15 THR 17 LEU 19 ARG 42 GLN 44 MET 46 SER 49 TYR 51 TYR 53 ASN 55 -49.450
TYR 2 GLN 4 TRP 6 ARG 8 HIS 13 GLU 15 THR 17 MET 19 TYR 42 GLN 44 GLN 46 SER 49 TYR 51 TYR 53 ASN 55 -49.160
GLN 2 MET 4 GLN 6 ASN 8 MET 13 ALA 15 TYR 17 LEU 19 ASN 42 LEU 44 MET 46 SER 49 ALA 51 TYR 53 MET 55 -48.680
TYR 2 GLN 4 GLU 6 ARG 8 GLU 13 ALA 15 LEU 17 LEU 19 TYR 42 THR 44 GLN 46 SER 49 ILE 51 TYR 53 TRP 55 -48.190
GLN 2 GLN 4 GLN 6 ARG 8 GLU 13 ALA 15 LEU 17 LEU 19 ARG 42 GLN 44 MET 46 SER 49 ILE 51 TYR 53 TRP 55 -46.540
ILE 2 MET 4 GLU 6 ASN 8 MET 13 LEU 15 TYR 17 ILE 19 ASN 42 LEU 44 ARG 46 ILE 49 ALA 51 ARG 53 MET 55 -46.010
ILE 2 GLN 4 GLU 6 ARG 8 GLU 13 GLU 15 LEU 17 MET 19 TYR 42 GLN 44 GLN 46 SER 49 ILE 51 TYR 53 TRP 55 -45.720
```

**Figure 2:** File with all the energies of all generations of mutants.

This file contains all the MA sequences of all generations sorted by energy, information necessary to generate new offspring. The task of generating new offspring is performed by the `crossing_over.c` script. From this point on, you can simply continue the project where we left off.

bash

```
g++ -o crossing_over.exe crossing_over.c
```

```
./crossing_over.exe
```

This uses `Population_classification.dat` to perform crossovers between the 10 mutants with the best energy. These crossovers result in the `crossing_over.dat` file. Once the new offspring are generated, we can execute the `generates_children.sh` script, which restarts the system to calculate the energy of these offspring.

bash

```
chmod +x generates_children.sh
```

```
./generates_children.sh
```

Then just execute `./genetic.sh` again to continue the GA cycle. Due to the limitation in the number of GPUs, we haven't developed a continuous GA cycle, so each generation takes approximately one week to be calculated. In this article, it took us approximately 6 months to complete the calculations.